Amendments to the Specification:

Please replace the paragraph beginning at page 2, line 29 with the following amended paragraph:

A computer program capable of generating digital data representing information communicated in a vertical blanking interval of a video signal may include a receiving module that receives data representing information communicated in a vertical blanking interval of an analog video signal and a generating module that generates digital data based on the data received by the receiving module. The generating module uses a predetermined algorithm to generate the digital data from the received data, which may represent non-video information. The received analog video signal may be, for example, a cable-broadcasted video signal, a satellite-broadcasted video signal, or a ~~telestial~~ terrestrial broadcasted video signal.

Please replace the paragraphs beginning at page 4, line 25 with the following amended paragraphs:

~~FIGS. 6A and 6B show examples of attributes and parameters for the user interface API of Fig. 5.~~

FIG. [[7]] 6 is a block diagram describing a data-loading API of the APIs of Fig. 4 for loading data into a database created using VBI transmitted data.

~~FIGS. 8A and 8B show examples of attributes and parameters for the data-loading API of Fig. 7.~~

FIG. [[9]] 7 is a block diagram showing an example of a mapping API of the APIs [[o]] of Fig. 4 for mapping channels and storing the mappings in a database created using VBI transmitted data.

~~FIGS. 10A and 10B show examples of attributes and parameters for the mapping API of Fig. 9.~~

FIG. [[11]] 8 is a block diagram showing an example of a system that receives, processes, and stores VBI transmitted data.

FIG. [[12]] 9 is a block diagram showing an example of the data flow between various modules of a system.

FIG. [[13]] 10 is a block diagram showing a data structure for storing generic tables in a database containing VBI transmitted data.

~~FIG. 14 is a block diagram showing an example of a memory management system for storing show information and video data.~~

FIG. [[15]] 11 is a block diagram showing an example of a system for updating files and database tables.

FIG. [[16]] 12 shows an example of a video signal containing VBI data.

Please replace the paragraph beginning at page 9, line 14 with the following amended paragraph:

The EPGChannelLineup function 501, which is supported by UI API 401 represents a query that searches for shows at a particular time. Aspects of one implementation of EPGChannelLineup function 501 are described in Table ~~FIG~~ 6A. The EPGChannelLineup function 501 may be called by a user interface (not shown) to display a list of brief show information for a given range of channels and a time frame. This function 501 receives, as input, a data structure including search criteria such as a start time, a stop time, a channel begin time, and a channel end time. This function 501 searches data store 302 based on the search criteria and builds an array containing information for each show including, for example, a channel number, a call letter or call letters of the channel, a start time, a duration, a category, a subcategory, a short title, and a reference number for the show. The function 501 returns TRUE if it is successful in performing the search and otherwise returns FALSE.

Please replace the paragraph beginning at page 9, line 25 with the following amended paragraph:

The EPGShowDescription function 502 represents another query supported by the UI API 401. This function 502 looks up a show description given a time and a show reference, and is described in greater detail with reference to Tables ~~FIGS.~~ 6A and 6B. This function 502 returns detailed show data including, for instance, the entire title, a short description, a full description, a category, a subcategory, the year produced, a television rating, a Motion Picture Association of America (MPAA) rating, an indication of the stars appearing in the show, and

several Boolean values indicating whether the show is a rerun, live, closed captioned, and/or stereo telecast.

Please replace the paragraph beginning at page 10, line 23 with the following amended paragraph:

Aspects of one implementation of the EPGFilter function 503 are described in detail with reference to ~~FIG.~~ Table 6B. This function 503 may be used to identify shows having a specified category-index and subcategory-index. The category-index and subcategory-index typically are numbered, for example, between 0 and 15; however, these limits can be changed to accommodate any number of categories.

Please add the following new paragraphs after the paragraph ending at page 11, line 6:

Table 6A

API for UI

1. EPGChannelLineup

Description Call this function to get a list of brief show information for a given range of channel numbers and time frame. The returned show array is sorted in the order of channel number and show beginning time.

Prototype Boolean EPGChannelLineup(const line_up & sline, show_info * const sinfo, int * const sorder, int &length, query_error & err);

Parameters •sline: IN PARAMETER. This structure specifies time grids and the range of channel numbers. The data structure is defined as:

```
struct line_up
{
    time_t time_grid_start;
    time_t time_grid_end;
    int channel _id_ start;
    int channel_id_end;
}
```

•sinfo: OUT PARAMETER. sinfo is an array of type show_info.

```
struct show_info
{
    int channel_number;
    char call_letter[9];
    time_t begin_time;
    short duration;
```

```
    unsigned char cat_index;
    unsigned char sub_cat_index;
    char short_title[22];
    long reference_number;

}
```

- sorder: OUT PARAMETER. It stores the order of shows sorted in channel number and time. For example, sinfo[sorder[0]] is the first show, sinfo[sorder[1]] the second, etc.
- length: IN OUT PARAMETER. As an in parameter, length is the dimension of sinfo and sorder, as an out parameter, it is the actual number of shows stored in sinfo and sorder, provided the number is less than the input length. Otherwise, FALSE is returned and err is populated with messages.
- Err: OUT PARAMETER. This parameter holds error information if error occurs. Non-zero err_code means something wrong in the call. Query_error is defined as

```
struct query_error
{
    int err_code;
    char err_msg[100];

}
```

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading

2. EPGShowDescription

Description This function returns a detailed show data. In order to get detailed show info, caller needs to pass the beginning time (or date) and reference number of the show. Reference number of a show is obtained by calling EPGChannelLineup function.

Prototype Boolean EPGShowDescription(show_brief_info &sbrief, show_description & sdesc, query_error & err);

Parameters • sbrief: IN PARAMETER. The struct show_brief_info is defined as

```
{
    time_t date;
    long reference_number;
```

Table 6B

```
}
```

sdesc: OUT PARAMETER. Detailed description of a show, including category, description, etc. The data structure is defined as

```
struct show_description
{
    char rest_of_title [100];
    char short_description[64];
    char description[801];
    char category[13];
    char subcategory[13];
    short year_produced;
    float stars;
    bool re_run;
    bool live;
    bool
    closed_caption;
    bool stereo;
    char tv_rating[13];
    char mpaa_rating[5];
    }
```

- err: OUT PARAMETER.

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

3. **EPGFilter**

Description To get a list of shows having specified category-index and subcategory-index. Category-index and subcategory-index are numbers between 0 to 15.

Prototype Boolean EPGFilter(show_ cat_ info &scat, show_info * const sinfo, int * const sorder, int &length, query_error & err);

Parameters • scat: IN PARAMETER. The data struct show_cat_info is defined as

```
struct show_cat_info
{
    short cat_index;
    short sub_cat_index;
    time_t begin_time;
    unsigned char updown_flag;
}
```

Above data structure specifies category index, subcategory index, beginning time of a search, and forward or backward search flag. updown_flag=1 means forward search, 0 means backward search. The search is limited among the shows of the same day as the specified beginning time.

- sinfo: OUT PARAMETER. sinfo is an array of type show_info.
- sorder: OUT PARAMETER. It stores the order of shows sorted in channel

number and time. For example, sinfo[sorder[0]] is the first show, sinfo[sorder[1]] the second, etc.

- length: IN OUT PARAMETER. As an in parameter, length is the dimension of sinfo and sorder, as an out parameter, it is the actual number of shows stored in sinfo and sorder, provided the number is less than the input length. Otherwise, FALSE is returned and err is populated with messages.
- Err: OUT PARAMETER. This parameter holds error information if error occurs. Non-zero err_code means something wrong in the call.

Return TRUE: Success

Value FALSE: Fails to prepare tables for data loading.

Please replace the paragraph beginning at page 11, line 7 with the following amended paragraph:

The UI API 401 described above provides a mechanism for building user interfaces for accessing an electronic programming guide. FIG. [[7]] 6 illustrates how the various functions interact with data store 302 to load data using the Data Loader API 402, including StartLoading function 701, WriteSIP function 702, and EndLoading function 703. The WriteSIP function 702 may use mapping array 704 which can be stored in memory. Functions provided in the Data Loader API 402 also are described in detail with reference to Tables ~~FIGS.~~ 8A and 8B.

Please replace the paragraph beginning at page 11, line 14 with the following amended paragraph:

The StartLoading function 701 prepares data tables in data store 302 to accommodate the loading of new or additional data. This function 701 may be called before data loading begins. The WriteSIP function 702 loads the data and detail information stored regarding each show, as described with reference to Tables ~~FIGS.~~ 8A and 8B. WriteSIP function 702 may use mapping array 704 to map channels to categories and vice versa. After all data has been loaded using the WriteSip function 702, the EndLoading function 703 may be called to commit the changes to the data store 302. This module may use transaction processing techniques to assure that the data store 302 is not left in an inconsistent state. When EndLoading function 703 is called, the system cleans up the data store 302 and updates all index and data tables in data store 302. By providing the StartLoading function 701 and the EndLoading function 702, the Data Loader API

402 allows information regarding many shows to be efficiently added to data store 302 while updating indexes and database tables a single time when the EndLoading function 703 is called. This greatly increases the efficiency of the data load process.

Please add the following new paragraphs after the paragraph ending at page 11, line 27:

Table 8A

API for Data Loader

1. **TsipCallback::StartLoading**

Description StartLoading is responsible for preparing database tables for the data loading. Therefore, this function must be called before data loading. After data loading is done, EndLoading function shall be called.

Prototype Boolean StartLoading()
Parameters None
Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

2. **TsipCallback::StartSip**

Description StartSip marks the beginning of the data loading of a show information packet. A show information packet contains 4-hour show data for one channel. EndSip indicates the end of the data loading of one packet.

Prototype Boolean StartSip()
Parameters None
Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

3. **TsipCallback::WriteSip**

Description WriteSip inserts one or couple of show data into database tables.
Prototype Boolean WriteSip(const SipHeader &shdr, const Showinfo *psi, int length)
Parameters • shdr: IN PARAMETER. SipHeader contains common data stored by show data stored in the array Showinfo.

```
struct
TUNECHAN {
   unsigned char type;    // b0-2: 0:OTA 1:cable 2:satellite 3-7:reserved //
                          b3:digital b4:dual A/B cable trunk b5-7:rsvd
   unsigned short minor; // up to 10 bits of digital minor channel
```

```
    unsigned short major; // up to 10 bits of digital major channel, or analog channel
}
struct SipHeader
{
    short channel_id;       // unique channel ID
     TUNECHAN tune_channel; // see struct definition above
    char channel_name[9];       // (null terminated string)
    unsigned char day_of week; // 0,1,2,6 for Sun, ..., Saturday, respectively

    char date[9];               // YYYYMMDD null terminated
                                // on 4 hour intervals (0, 4, 8, 12, 16, 20)
}
```

• psi: IN PARAMETER. Showinfo is defined as

```
struct Showinfo
{
    char short_title[22];    // short title, as displayed on grid titles
    char rest_of_title[100];        //concatenated with short title, the complete show title
    char short_description[64]; // short description. Enough descriptive text to fit 3x21
                                // display. (includes ratings for movies, teams for sports,
                                // subjects for talk shows, etc. char null terminated.
    char long_description[801]; // with short description, complete description
                                // null terminated
    char begin_time[ 15 ];
```

Table 8B

```
        // Show starting time with format of YYYYMMDDHHMI
        // null terminated. MM takes a value of 01, ..., 12;
        // DD takes a value of 01, ..., 31; HH takes a value of 00, ..., 23;
        // MI takes a value from 00 to 59.
    char end_time[15]; // as above, used for performance purpose
    int duration; // in minutes
    char category[13]; // category name, null terminated
    unsigned char cat_index;
    char subcategory[13]; // sub-category name, null terminated
    unsigned char sub_cat_index;
    short year_produced;  // four digits, e.g., 1998. For search purpose
    float stars;              // 0, 0.5, 1, 1.5, 2, ..., 4.5, 5 for search purpose
    bool re_run;
    bool live;
    bool closed_caption;
    bool stereo;
    char TV_rating[13]; // "TVMA-FVDLVS", "TVY-LV", ..., null terminated
    char mpaa_rating[5]; // "NC17", "X", ..., null terminated
};
```

psi is the pointer of the array Showinfo psi[length].
• length: IN PARAMETER. length tells how many show records in the array psi.

Return TRUE: Success
Value FALSE: Fails to prepare tables for data loading.

4. TsipCallback::EndSip

Description EndSip marks the end of the data loading of a show information packet.

Prototype Void EndSip()
Parameters None **Return**
Void **Value**

5. TsipCallback::EndLoading

Description After data loading is successfully completed, Data Loader calls EndLoading to invoke a sequence of actions on the database tables, such as clean up temporary tables and etc. If, for some reason, this API is not called, then show data for UI will not be updated.

Prototype Void EndLoading()
Parameters None **Return** Void
Value

Please replace the paragraph beginning at page 11, line 28 with the following amended paragraph:

With reference to FIG [[9]] 7, an example of the Mapping API 403 provides functions for retrieving and setting mapping data for the system, such as for example, Get/SetChannelMap 901 and Get/SetMiniGuide 902. The data store 302 generally includes two tables for storing mapping data: Channel Mapping table 903, with about 100 rows, and Mini Guide table 904, with about 15 rows. Functions 901 may be called to read or update the Channel Mapping table 903. Examples of these functions 901 are described in ~~FIG~~ Table 10A as MapGet and MapSet.

Please replace the paragraph beginning at page 12, line 15 with the following amended paragraph:

Similarly, the Mini Guide table 904 contains information regarding the mapping between channels and categories. The MiniGuideGet and MiniGuideSet functions 902 provide a mechanism for viewing and modifying data stored in the Mini Guide table 904. Aspects of one implementation of these functions 902 are described with reference to Tables ~~FIGS.~~ 10A and 10B.

Please add the following new paragraphs after the paragraph ending at page 12, line 19:

Table 10A

API for Channel Mapping

1. MapGet

Description This function returns an array of Channel Mapping data to the caller. Channel Mapping describes the mapping among Channel, Call letter, Category, and Channel Id.

Prototype Boolean MapGet(channel_map * const minfo, int * const sorder, int &length, query_error & err)

Parameters • minfo: OUT PARAMETER. The pointer minfo points to an array of channel_map. The struct channel_map is defined as

```
struct channel_map
{
   char call_letter[9];
   unsigned short channel id;
   unsigned short channel;
   unsigned short cat;
};
```

- sorder: OUT PARAMETER. It stores the order of Channel Mappings sorted in channel number.
- length: IN OUT PARAMETER. As an in parameter, length is the dimension of minfo and sorder, as an out parameter, it is the actual number of shows stored in minfo and sorder, provided the number is less than the input length. Otherwise, FALSE is returned and err is populated with messages.
- Err: OUT PARAMETER. This parameter holds error information if error occurs. Non-zero err_code means something wrong in the call.

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

2. MapSet

Description Caller can set Channel Information by calling this function. Inside of MapSet, it checks each element of the array minfo against the records in the database. If the call letter exists in the database, then update the row by the new data from minfo; if call letter doesn't exist in the database, then insert a new row into the database.

Prototype Boolean MapSet(channel_map * const minfo, int &length, query_error & err)

Parameters • minfo: IN PARAMETER. The parameter minfo is an array of channel_map. The struct channel_map is defined as

```
struct channel_map
{
    char call_letter[9];
    unsigned short channel id;
    unsigned short channel;
    unsigned short cat;
};
```

- length: IN PARAMETER. It is the actual number of Channel Mappings stored in minfo.
- Err: OUT PARAMETER. Non-zero err_code means something wrong in the call.

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

3. MiniGuideGet

Description To get Mini Guide data. Mini Guide shows the mapping between Category and Channel. At present, there are about 12 categories. Each Category owns a segment of Channels.

Prototype Boolean MiniGuideGet(mini_guide * const minfo, int * const sorder, int &length, query_error & err)

Table 10B

Parameters • minfo: OUT PARAMETER. minfo is an array of mini_guide. The data struct mini_guide is defined as

```
struct
{
  char name[15];                 // Category
  unsigned short channel;        // Channel Number
  unsigned short offline;        // Channel number starts at
  unsigned short start;          // Channel segment starts at
  unsigned short end;            // Channel segment ends at
  unsigend short dup_start;      // Local Channel starts at
```

};

- sorder: OUT PARAMETER. The order of Mini Guides by channel number.
- length: IN OUT PARAMETER. As an in parameter, length tells the dimension of minfo and sorder, as an out parameter, it is the actual number of Mini Guides stored in minfo and sorder, provided the number is less than the input length. Otherwise, FALSE is returned.
- Err: OUT PARAMETER. Non-zero err_code means something wrong in the call.

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading

4. MiniGuideSet

Description Caller can set Mini Guides by calling this function. Inside of MiniGuideSet, each element of the array minfo is checked against the records in the database. If a category exists in the database, then update the row by the new data from minfo; if a category doesn't exist in the database, then insert a new row into the database.

Prototype Boolean MiniGuideSet(mini_guide * const minfo, int &length, query_error & err)

Parameters

- minfo: IN PARAMETER. The pointer minfo points to an array of mini_guide.
- length: IN PARAMETER. It is the actual number of Mini Guides stored in minfo.
- Err: OUT PARAMETER. Non-zero err_code means something wrong in the call.

Return Value TRUE: Success
FALSE: Fails to prepare tables for data loading.

Please replace the paragraph beginning at page 12, line 20 with the following amended paragraph:

With reference to FIG [[11]] 8, a system is described for receiving and processing data from video signals in which the Data Loader API 402 uses data that has been received by receiving module 307 and processed by generating module 308.

Please replace the paragraph beginning at page 12, line 28 with the following amended paragraph:

More specifically, in one implementation, a video signal 1101 is received, sampled, and digitized into a digital value array 1102 by driver 1103. This array 1102 is converted into a character string by converter 1104. An API may be used to enable access to the data stored in the character stream. The accessed data is stored in memory buffer 1105, where it can be accessed

by the Data Loader API 402 and stored in data store 302. The Data Loader API 402 accesses the data store 302 using a communications interface 1106. Commercial databases usually include several communications interfaces that permit access to a database by an application running on the same machine as the database and by an application running on a different machine connected via a computer network. Once the data store 302 has been populated, various APIs 1107 such as the UI API 401, the Mapping API 403, and other APIs 404 may be used to access the data. FIG. [[11]] 8 shows an EPG UI 1108 that uses an API 1107 (such as those described in FIG. 4) to access data.

Please replace the paragraph beginning at page 13, line 9 with the following amended paragraph:

With reference to FIG. [[12]] 9 illustrates an exemplary data flow among several modules (e.g., a User Interface Module 1202, a Data Loader Module 1203, and a Data Management Module (DMM) 1201) and various files 1204. The DMM 1201 may be a general purpose database, or it may be designed as a specialized database as described herein for managing EPG data. For example, a special-purpose database may be designed that provides a high level of performance in a simple, stable, and small implementation with the ability to handle large volumes of data.

Please replace the paragraph beginning at page 13, line 28 with the following amended paragraph:

Referring to FIG. [[13]] 10, in DMM 1201, a table may be included in a file that stores formatted data. The file has two major data areas: a header area 1301 that contains managerial data, and a body area 1302 holds the data of the table. Further, the body area 1302 may be formed by rows with fixed lengths, with each row having its own row-header and row-body.

Please replace the paragraph beginning at page 14, line 9 with the following amended paragraph:

Similarly, a row-header may be unique for every row and every table, containing managerial data indicating a row identifier, a flag indicating whether the row is use, and the next

available row. A row-body is determined by data stored in the table. Different tables have different row-body definitions. For example, ~~FIG.~~ Table 14 describes an exemplary implementation of a structure for a table header and a row header.

Please add the following new paragraphs after the paragraph ending at page 14, line 13:

Table 14

```
struct adr_hdr;
{
        char_name[30];
        char f_name[30];
        char c_date[15];
        long t_row;
        long a_row;
        long na_row;
        short hdr_len;
        short row_start;
        short row_len;;
        short row_hdr_len;
        short row_bdy_len;
        short row_hdr start;
};
```

where

- Name: Table name
- C_date: When the table is created, in the form "YYYYMMDDHHMISS"
- F_name: the file where the data is stored
- T_row: Total number of initialized rows
- Na_row: Next available row id for insertion
- Hdr_len: length of the header
- Row_start: where the data (row) start (=hdr_len);
- Row_len: the length of a row
- Row_hdr_len: length of the row header
- Row_bdy_len: the length of the row-body
- Row hdr start: Where the row-header starts

```
Struct row_hdr
{
```

```
        long row_id;
        long next_available_row;
        char usage;
    }
```

where

- Row_id: row id of the row. Increased by one each time
- Next_avaialble_row: row id of the next available row for insertion.
- usage: 'N' means available; 'Y' means used.

Please replace the paragraph beginning at page 14, line 14 with the following amended paragraph:

FIG. [[15]] 11 illustrates a system including call-back functions to assist in preserving RAM space. In one implementation, video data and show information data are separated and stored on a hard disk or other computer readable medium. A group of call-back functions are provided to store decoded show information into files or database tables. FIG. [[15]] 11 illustrates how the Data Loader API 402 may operate to access this data. The system maintains a temporary database table 1501 of show programming information for an electronic programming guide. When the system calls the StartLoading function, the temporary database table 1501 is created. Calls to WriteSip instruct the system to write to the temporary database table 1501. When the EndLoading function is called, the temporary database table 1501 is used to update the actual database table 1502 and the indexes are updated accordingly. Users accessing the primary database may not have access to the updated data until the update is complete.